A Study of MAPSE Extensions

David Auty, SofTech, Inc.
Robert Charette, SofTech, Inc.
Charles McKay, UHCL High Technology Laboratory

1. Overview

This project was initiated to study the technical issues of extending the MAPSE to support the life cycle of large, complex distributed systems such as the Space Station Program (SSP). The work has been divided into two phases. Phase one, covered by this report, identifies a list of advanced technical tools needed to extend the MAPSE to meet the needs believed to be inherent in the Software Support Environment (SSE). Of secondary importance was the identification of a list of advanced management tools.

Phase two, which is on-going at this time, is to study and document the major technical issues in adding these tools to the MAPSE as an integrated extension evolving into an appropriate SSE. The intent is to provide a framework for understanding and evaluating the subsequent development and/or procurement of such tools.

This paper has been extracted from the full interim report on the phase one efforts. It incorporates just the description of SSE requirements, and a list of the tools identified. Other topics addressed in the interim report include an outline of the principle requirements for a MAPSE, a description of the life cycle model and a description of the tools in the context of the life cycle model.

For the purpose of this paper, the basis life cycle model is an adaptation of the symbolic representation of McDermid and Ripken (1984) to the model described in DoD Standards 2167 and 2168. The model partitions the process of software development into the following phases:

p1: System Requirements Analysis,
p2: Software Requirements Analysis,
p3: Preliminary Design,
p4: Detailed Design,
p5: Coding and Unit Test
p6: Computer Software Component Integration

The outputs from each phase are the formal review documents used for verification and validation, which also form the inputs to the succeeding phases. All documents and development information are maintained in an integrated life cycle project object base which serves to centralize and control the development process. All activities and tools work with this project object base to maintain the parallel processes of configuration and quality control.

2.0 A Brief Description of Support Environment Requirements in the Context of the Life Cycle Model

2.1 System Requirements Analysis

2.1.1 Characteristics, Principles and Methods

Several activities should be pursued during requirements interpretation, feasibility studies, and analysis.

Semantic Information Capture - Supporting interpretation, the capture of requirements in the form of a semantic model involves identifying key terms, categorizing the terms, defining the terms, and identifying the relations between the terms. The capture of semantic information creates a recording of the semantic model of the requirements, which becomes part of the baseline. Assuming the semantic information is machine-encoded, it might be expressed in a formal language such as Problem Statement Language (PSL) or in a combination of formal graphics and text expression such as Software Requirements Engineering Methodology (SREM).

Semantics Analysis - Once the requirements are expressed in the context of a semantic model, the model relations can be used for a systematic analysis of the completeness and consistency of the requirements. This is achieved by asking questions which are answered with the aid of the relations, such as "Are there any other processes which should be related to Process A by the 'predecessor of' relation?"

Traceability may be established through reference relations between requirements and specification, design and code, etc. The relational analysis can be used to assess the impact of requirements changes on the baselined products.

The semantic analysis activity aids development by identifying areas of requirements incompleteness or inconsistency.

Feasibility and Risk Analysis - Evaluating the feasibility of requirements is a significant part of requirements analysis. Feasibility should be viewed from the perspectives of design, performance and cost.

Design feasibility involves finding at least one design that satisfies the requirements. Any approach from trial design to prototyping is appropriate. Performance feasibility is a special case of design feasibility analysis. Once a trial design is established, modeling is an effective technique for analyzing performance. Cost feasibility involves estimating costs based on the trial design. Cost analysis must consider the three key elements: the development phase, the operations phase, and the phase for continuing adaptation.

2.1.2 Requirements on the Support Environment

The requirements on the support environment data base, derived from requirements analysis, are:

Baselined Products - The semantic information is the only data associated with requirements analysis that should be baselined. It should be under configuration control and subject to change only as requirements changes are approved. Baselined data should not only include the "shalls" of each phase (which must be dichotomously demonstrated at acceptance test time) but also the "shoulds" which have life cycle implications that cannot be dichotomously demonstrated at acceptance test time and which may require the design of special metrics and instrumentation to support their analysis at subsequent points in the life cycle.

Non-Baselined Data - Any information associated with modeling, simulation, prototyping, or semantic analysis should be saved temporarily. It should be used later in requirements analysis iteration or other activities.

Measurement Data - Several measurements of the requirements analysis activity and its outputs should be captured:

- Size of the data base for semantic information,
- Complexity of the requirements as measured by the relationships in the semantic information and
- Number of inconsistencies or omissions found.

2.2 Software Requirements Specification

2.2.1 Characteristics, Principles and Methods

Formal Recording - The specification information must be recorded in some suitable form. As a minimum, the specification should describe interactions, modes and functions. It should have the characteristics of being minimal, understandable, accurate and precise, and easily modified. The specification should use a formal notation to facilitate formal correctness analysis and automated analysis of the specification easier.

Completeness Analysis - This is done by trying out a design of the system. Completeness analysis generates questions which can help identify information absent from the requirements. In many cases, this activity is done during requirements analysis.

Correctness Demonstration - The specification must be shown as consistent with the requirements. Since the requirements may not be stated in a formal manner, a rigorous proof of their consistency may not be possible. The correctness demonstration is then produced through a subjective, informal analysis based on the semantics information from requirements analysis.

Consistency Analysis - Any method for performing this analysis depends on the form of the specification information. If a formal specification language is used, certain kinds of problems may be detected by analyzing this notation. In other cases, the consistency of the specification information must be judged on a less precise basis. A good example of the state-of-the-art in specification methods is that advocated by the Naval Research Laboratory, and used to develop the specification of the A-7 flight program. The specification document includes formal, tabular notation which lends itself to completeness and consistency analyses.

2.2.2 Requirements on the Support Environment

The requirements on the support environment data base, derived from the specification activity are:

Baselined Products - The specification information is baselined. Any modeling information produced should be baselined if it is crucial to the life cycle support of the software.

Non-Baselined Data - This material includes partial specifications under development, alternate specification, and diagnostic information produced by specification analysis tools.

Measurements - Examples of useful measurements data to be captured are: effort and resource data concerning the development of the specification, size data, number of errors and changes made, and subjective measures of the quality and completeness of the specification.

2.3 Preliminary and Detailed Design

2.3.1 Characteristics, Principles and Methods

A design is the translation of the "shalls" from requirements analysis into Ada package specifications. Functional requirements should be transformed into functional Ada specifications that can be checked by an Ada Compiler. Non-functional requirements (i.e., constraints) should be transformed into a discipline of Ada comments that can be checked by other APSE tools.

Three areas of design support are identified: formal recording of system design, formal recording of data and program design, and creative aids.

Formal Recording of System Design - There are several methods involved in recording the system design.

Information-Hiding - This method involves isolating information within modules. The module limits are defined by the information (design decisions, data definitions, etc.) to be isolated. Design is based on the expected changes to the information, thus localizing the effect of future changes.

Module Specification - Focusing on module specifications yields a description which allows others to determine the intent of a complete module by reading the module specification.

Use Hierarchy - Focusing on the use hierarchy yields a description which explains which programs depend on the correct implementation of a given module to produce correct results.

Formal Recording of Data and Program Design - The techniques and methods for the formal recording of data design and program design are:

Program Design Language (PDL) - The writing of data and program design in a PDL is a useful technique for formally recording the program design. It is sufficiently low-level to support direct coding, and is flexible enough to leave some questions unanswered while the designer proceeds with the design. (i.e., Ada Source code with Ada "stubs".)

Stepwise Refinement - This method goes hand in hand with PDL. With stepwise refinement, specifications for the lower level code become part of the documentation of the procedure. This makes the intent of the code much clearer.

Abstraction of Data Types - With abstraction, the designer can develop details where they are needed. This permits information-hiding as well as a more independent implementation of the system.

Creative Aids - Many creative techniques exist for design. A designer chooses techniques based on their individual approach to creativity. Some prefer graphic techniques while others do not. The choice of creative techniques should be left to the individual, whereas the techniques for formal recording must be standard. Described below are some representative creative aids:

Data and Control Flow Analysis - Module decomposition and function allocation are based upon the data and control flows required by the system. An example is Structured Design.

Data Structure Transformation - Transformation is a design technique in which the structure of the input and output data determines the structure of the program.

Graphic Decomposition Techniques - Graphs showing hierarchic relations depict the decomposition at many levels. An example is Structured Analysis and Design Technique (SADT).

Graphic Control Descriptions - Other ways of showing the control flows in the program are Petri Nets and Warnier-Orr diagrams.

2.3.2 Requirements on the Support Environment

As with the other activities of development, the data base must contain information on the design.

Baselined Products - Throughout the life of the system, the most recently approved form of the design must be stored in the data base. The system design is entered before the design of various subsystems or modules.

Non-Baselined Data - This includes preliminary designs as well as graphic displays used during the creative process. Graphic displays include tree structures, block diagrams, and other material created by design tools. The data base must provide for maintaining the temporary designs developed before one is actually chosen and baselined.

Measurements - These should include module interconnection measurements, such as data bindings. These should also include lower design measurements, such as cyclomatic complexity, and operators and operands. Many of these measurements are normally taken on the completed code, but with good, low-level PDL, they can be taken (or approximated) during design.

Archival Data - Archived data should capture the motivation behind the choice of design. The archived data should also include past designs evolved from use or rejected during development along with the reasons for the rejection.

2.4 Coding and Unit Test, Computer Software Component Integration

2.4.1 Characteristics, Principles and Methods

This section will focus on the unique requirements of developing distributed systems.

Designs which map program entities across distributed processing resources should be specified in two complementary parts. First, the functional requirements should be demonstrated to be met by the program design by executing the program in the host environment. (I.e., compile and execute the Ada source code on the host system without regard to properties of distribution.) Second, the non-functional requirements (i.e., constraints) such as the location each program entity is to be assigned, timing constraints, sizing constraints, etc. should be mapped to a simulator for analysis of the implications of imposing these restrictions upon the design which was proven in the first step. Tuning of assignments, code, algorithms and structures can take place in the host environment until the simulator provides a degree of confidence. Load modules can then be built and moved to the target environment or to a target test bed for further study. The implementation should produce an effective, understandable transformation of the design. The automatic generation of appropriate comments in the source code can ease the more complex process of maintenance in a distributed environment.

The following are some key aspects of implementation:

Standard Interface Set to a Catalog of Runtime Support Environment Features and Options. - This interface set establishes a virtual Ada machine. The compilation system produces target code that uses the services provided by the standard interface set. The requested service determine which modules of the runtime support library are to be exported to the target environment.

Target Network Topology Specification - This allows the designer to specify the symbolic names for remote area networks, local area networks, and individual processing nodes. The design also identifies the communications support available to link the various entities of the network.

Target Node Resources Specification - This allows the designer to specify the hardware resources for each node identified with the network topology specifier. The system will retain this information in the project object base along with the collection of software resources that will be assigned to this node later in the design. The designer declares the instruction set architectures available, the memory banks and their attributes, the buses and their attributes, and the communications links that are available.

Partitioning and Allocation Specification - After the Ada source code has been transformed into a DIANA representation and executed to demonstrate that it meets the functional requirements of the program, a discipline of comments and key words such as "location" can be used to map each program entity to a symbolic location. This symbolic location corresponds to those node and network identifications previously entered with the topology specification and the node resources specification. These non-functional requirements are added as attributes to the DIANA representations.

Distributed Workload Simulation - After the symbolic location assignments and other constraints have been added to the attributes of the DIANA representation, the workload simulator examines the project object base to determine characteristics of the already existing workload (if any) and to select empirical estimates of communications delays, processing throughput, and other relevant estimators. A simulation is then provided for analysis. If the analysis indicates the design approach is not feasible, new approaches to distribution can be provided by returning to the partitioning and allocation specification.

Distributed Program Building - When the workload simulation indicates a feasible design, the process of building new load modules includes examining the symbolic location assignments added to the DIANA tree and looking these up in the project object base to determine what type of instruction set architecture the particular entity's object code is to be generated for. If the code is to be added to the workload of an existing system, it is also necessary to identify if additional modules or new versions of the run time library need to be added or if additional hardware is likely to be needed to accommodate the increase in workload. The end result of the program building activity is to prepare a load module consisting of applications code and the necessary support from the run time library for each of the processors affected by the distribution of the program entities.

Run Time Support Environment Monitoring - If life and property are to depend upon the program meeting both its functional and its non-functional requirements, it may be desirable to prepare the program for execution in a target testbed. to be effective, the testbed should be fully instrumented and interact with the host environment. This requires the support of a run time monitor for each processor in the target testbed to interact with the instrumentation and host environment to provide meaningful information.

2.4.2 Requirements on the Support Environment

The most important requirements and opportunities for the support envrionment life cycle project object base become evident from this phase. The results are summarized below:

Baselined Products - The functional requirements are similar to those described in the preceding sections. However, opportunities arise due to the requirements for the DIANA representation in the implementation phase. An estimated ten to twenty times the processing time is required to convert Ada source code to DIANA representation as compared to converting the DIANA representation to object code for the target environment. Furthermore source

code and object code can both be reconstructed from the DIANA representation. Since the Stoneman requirements for the MAPSE provides a unique identification for each object produced (which includes history attributes identifying the time, date, tools, etc. used to manipulate the object), an enormous amount of on-line storage space can be conserved in the project object base if the DIANA representation is maintained as the baseline.

The other important implication for baseline control as a result of this phase is the identification and maintenance of the network topology and the network node resources described in the preceding section

Non-Baselined Data - The temporary storage required for this category is similar to the functional requirements listed in the other sections of this report. However, the savings and storage space made possible by the utilization of DIANA representation described above may be significant even for temporary storage requirements.

Measurement Data - A number of metrics regarding the utilization of these tools is desirable. Knowing who is using the tools for what projects, and knowing the frequency of reference can provide valuable management insights.

2.5 Verification and Validation

2.5.1 Characteristics, Principles and Methods

The methods linked with correctness analysis are either static analysis or dynamic analysis. Static analysis includes, in order of increasing rigor, reviews, inspections, and proofs of correctness. Dynamic analysis includes all testing techniques.

Reviews - Reviews determine the internal completeness and consistency of system requirements and software specification, design and test information. They also assess its consistency with its predecessor information. Reviews involve a broad range of people, including developers, managers, users, and outside experts or specialists. A review must have specific objectives and questions to be addressed. The review findings generate rework tasks for the development group.

Inspections - Inspections evaluate the correctness of component level specification, design, code, test plans, and test results. They are more formal and rigorous than reviews. An inspection involves a small group of people of a specific make-up, and follows a well-defined procedure.

Proofs of Correctness - All development products should be verified with an informal proof of correctness. Certain critical kernels of code or special applications may require a formal proof of correctness.

Testing - Dynamic execution of the system or system component with known inputs in a known environment is a "test". If the test result is consistent with the expected result, the component is deemed correct in the limited context of the test. The following baselined documents are created relative to testing:

- Test Plan - Defines the scope, approach, and resource needed for testing.

- Test Procedures - Provides a detailed description of the steps and test data associated with each test case.

- Test Results - Documents the results of each test run. Unsuccessful runstrigger trouble reports which must be addressed by the development group.

The relationships between system functions and component or system test cases should be clearly established. Then, when changes are made to parts of a system, a subset of test cases can be identified which will test the system sufficiently. This process is called regression testing. Effective regression testing is a good way to reduce software development costs.

2.5.2 Requirements on the Support Environment

The requirements on the Support Environment data base, derived from the correctness analysis, are summarized below:

Baselined Products - Test plans, test procedures and test results (of correctly executed tests) are all baselined. They are controlled by configuration management. The results of inspections and proofs might also be baselined.

Non-Baselined Data - The non-baselined data includes work-in-progress, static analysis data, trouble reports, and debug data. Temporary storage of this type of information is required.

Measurement Data - A number of measurements associated with correctness analysis should be captured. These include: number of modifications to a unit, number of errors found per unit, number of test runs, number of errors by error category, and test coverage.

2.6 Project Management Support

2.6.1 Characteristics, Principles and Methods

Estimation - Most resource estimation techniques use the measurements from prior projects to estimate resources. Support of estimation methods requires a data base of comprehensive measurements including such software system parameters as size of source code, source language, development resources expended, and complexity measures.

Precedence Networks - This planning method is used to analyze task dependencies and to determine the critical path of development activities. Such an analysis is usually needed to define a realistic schedule. It is also useful in evaluating contingencies and creating contingency plans.

Change Control - This is the core of configuration management. It controls all changes to baselined products. The approval process for changes might be as follows:

- The written request for change is submitted to the configuration management function. It might come from a change in requirements or from a trouble report documenting a defect.

- An assessment is made of the technical feasibility of the change, and its impact on schedule and budget. If it has the potential to endanger life and property, a separate safety assessment may be made.

- The change is approved or disapproved based on its potential effect upon safety, its value and its cost.

- The development plan is modified and resources adjusted to add approved changes.

- The fully verified change is entered into the new baseline.

2.6.2 Requirements on the Support Environment

The activity of management imposes the following requirements on the support environment data base.

Baselined Products - The development plan, although not a part of the software system or its descriptive information, should be maintained as a baselined product to insure proper management of changes to the plan. Configuration management data and quality assurance plans should also be baselined.

Non-Baselined Data - Significant amounts of information associated with the management must be kept temporarily. This information includes engineering change requests, trouble reports, resource allocation plans, actual resource utilization reports, technical milestone status, action item status, and the results of quality assurance reviews.

Measurement Data - Many measurements are of interest to management. These include the number of engineering change proposals (ECP), and trouble reports (TR), time to process an ECP or TR, resource use for each ECP or TR, resource use by project activity, and software size and complexity measures.

3.0 Tools to Extend the MAPSE

Data Entry

Problem Expression Editor (for requirements analysis, specification)
Syntax/Template Directed Editor
Menu Manager
Graphics Package (GKS, 2D, 3D)
Word Processing integrated with Graphics and Electronic Mail
Network communications across hosts and targets

Library Aids

Semantics Information Browser
Diana Tree Browser
Reuseable Components Browser
Dictionary and Schema Tools
Host CLP script Manager

Management Aids

Report Generator
(Integrated Text and Graphics Forms Generator)
Change Request Tracker
Automated Precedence Network
Resource Scheduling Aid
Automated Work Breakdown Structure
Event Flags&Signals Generator
(signal path planning)
Schedule Generator

Syntax/Semantics Analysis

Requirements Language Processor
Consistency/Completeness Checker
Requirements Information Analyzer
Design Specification Language Processor
Standards Checker
PDL Syntax Analyzer
Requirements to Design Tracer/Checker
Design Complexity/Metrics Analyzer

Proof/Assertion Checker

Verifyer/Assertion Analyzer
Theorem Prover
Symbolic Execution System

Implementation Support

Compilation Order Analysis
Call Tree Report Generator
Automated Recompilation
Performance Metrics Analyzer
Elaboration Dependencies Analyzer
Cross-Reference Generator
Change Control and Impact Assessment
Statement Profile Generator
Generic Usage Report Generator
Diana Tree Expander

Test Generation, Analysis, Automation

Test Harness
Generic Instantiation Harness
Test Data Generator
Black Box Test Generator
Data Extraction and Reduction
Test Results Comparator
Target System Testbed (fully instrumented)
Environment Simulator/Stimulator
Performance Monitor
PDL Interpreter
Test Coverage Analyzer
Test Completeness/Consistency Analyzer
Target Emulation/Simulation
Scenario Generator
Fault Stimulator/Analyzer

Modeling/Simulation

Resource Estimator
Modeling Tool
Prototyping/Simulation Capability
Performance Model
Reliability Model

Run-Time System Support

Runtime Support Dependencies Analyzer
System Timing Analyzer
System Tasking Analyzer
Runtime Monitor
System Storage Analyzer

Distributed Target System Support

Target Node Resources Editor
Target Network Topology Editor
Partitioning and Allocation Editor
Distributed System Generator (program builder)
Distributed workload simulator

Expert Systems

Real-Time Assistant
Fault-Tolerance Assistant
Reuseable Components Assistant
Upgrade Load, Test and Integration Planning Aid (for non-stop nodes)
Expert System Generator